Chapter 5

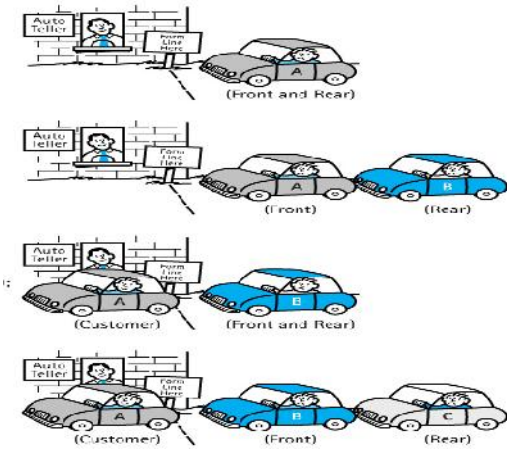
ADTs Stack and Queue

Outline

- Stack
  - Array-based Implementation
  - Linked Implementation
- Queue
  - Array-based Implementation
  - Linked Implementation
- Comparison

Queues

Queues



Queues

•*What do these composite objects all have **in common**?*

Queues

Queue

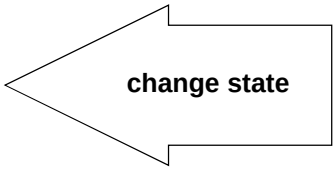
An abstract data type in which elements are added to the rear and removed from the front; a “first in, first out” (**FIFO**) structure

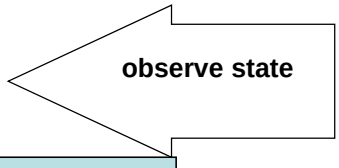
Queues

- *What operations would be appropriate for a queue?*

Queues

- **Transformers**
  - MakeEmpty
  - Enqueue
  - Dequeue
- **Observers**
  - *IsEmpty*
  - *IsFull*





What about an iterator?

Queues

- *For what type of problems would queues be useful?*
  - ***Print server***
    - *maintains a queue of print jobs.*
  - ***Disk driver***
    - *maintains a queue of disk input/output requests.*
  - ***Scheduler (e.g., in an operating system)***
    - *maintains a queue of processes awaiting a slice of machine time.*

Queues

Logical Level

```
class QueueType
{
public:
  QueueType(int max);
  QueueType();
  ~QueueType();
  bool IsEmpty() const;
  bool IsFull() const;
  void Enqueue(ItemType item);
            //add newItem to the rear of the queue.
  void Dequeue(ItemType& item);
                //remove front item from queue
```

Array-Based Implementation

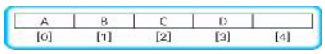
```
private:
  ItemType* items; // Dynamically allocated array
  int maxQue;
  // Whatever else we need
};
```

Implementation level

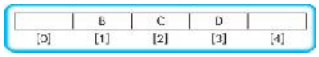
Array-Based Implementation

One data structure: An array with the front of the queue fixed in the first position

Enqueue A, B, C, D



Dequeue

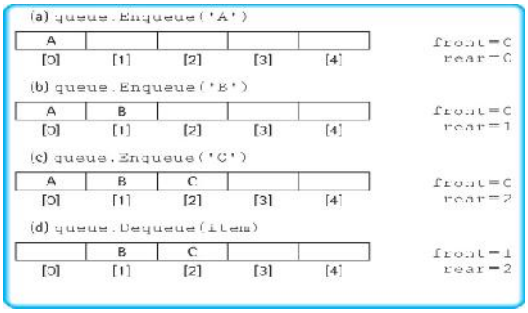


Move elements forward

What's wrong with this design?

Array-Based Implementation

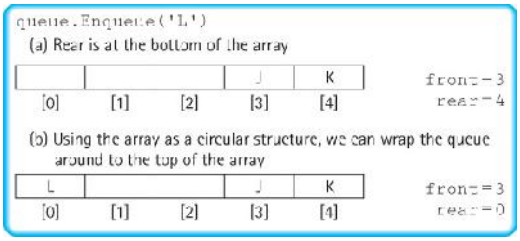
Another data structure: An array where the front floats



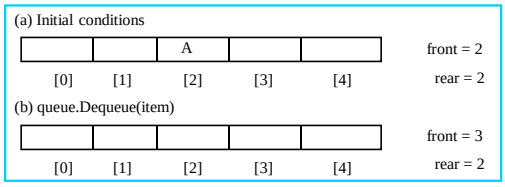
What happens if we add X, Y, and Z ?

Array-Based Implementation

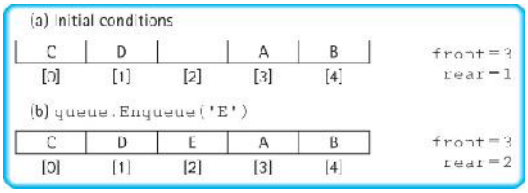
We can let the queue wrap around in the array; i.e. treat the array as a circular structure



Array-Based Implementation



Empty Queue

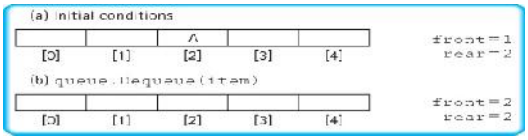


Full Queue

How can we tell the difference?

Array-Based Implementation

A third data structure:Front indicates the slot preceding the front item;it is reserved and not used



Empty Queue

Full Queue

Array-Based Implementation

```
private:
  int front;
  int rear;
  int maxQue;

  ItemType* items;
};
```

Complete implementation level

*To what do we initialize **`front`** and **`rear`** ?*

Array-Based Implementation

```
QueueType::QueueType( int max)
{ maxQue = max + 1;
  front = maxQue - 1;
  rear = maxQue - 1;
  items = new ItemType[maxQue];
}

QueueType::QueueType( )
{ maxQue = 500 + 1;
  front = maxQue - 1;
  rear = maxQue - 1;
  items = new ItemType[maxQue];
}

QueueType::~QueueType( )
{
   delete [] items;
}
```

*Why is the array declared **`max + 1`** ?*

Array-Based Implementation

```
//Pre: the queue is not full
//Post: newItem is at the rear of the queue
void QueueType::Enqueue(ItemType newItem)
{
  if (IsFull())
     throw FullQueue();
  else{
      rear = (rear+1)%maxQue;
      items[rear] = newItem;
  }
}
```


Array-Based Implementation

```
//pre: the queue is not empty
//post: the front of the queue has been removed
// and a copy returned in item
void QueueType::Dequeue(ItemType& item)
{
  if (IsEmpty())
      throw EmptyQueue();
  else{
     front = (front+1)%maxQue;
     item = items[front];
  }
}
```

Array-Based Implementation

```
//returns true if the queue is empty
bool QueueType::IsEmpty( )
{
  return ( rear == front );
}

//returns true if the queue is full
bool QueueType::IsFull( )
{
  return ( (rear + 1) % maxQue == front )
}
```


Counted Queue

Counted Queue

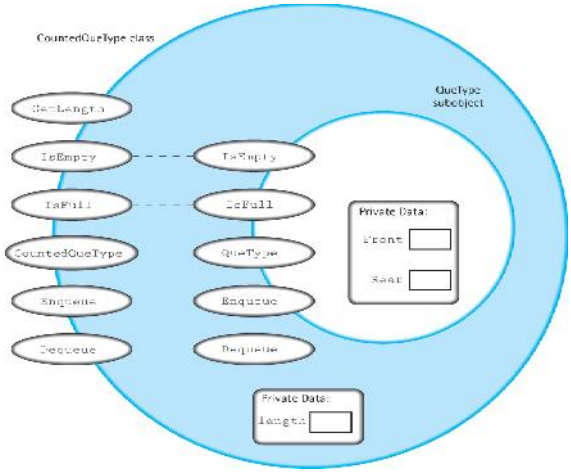
A queue that has an additional function **`GetLength`** that returns the number of items in the queue

What functions must be defined in class ***`CountedQue`****?*

Counted Queue

```
class CountedQueueType : public QueueType
{
public:
  CountedQueueType();
  void Enqueue(ItemType newItem);
  void Dequeue(ItemType& item);
  int GetLength( ) const;
private:
  int length;
};
```


Counted Queue



Counted Queue

```
void CountedQueueType::Enqueue(ItemType
newItem)
{
  try
  {
    QueueType::Enqueue(newItem);
    length++;
  }
  catch(FullQueue)
  { throw FullQueue();}
}
```

What is this?

*Why must the call to **Enqueue** be embedded within a try/catch statement?*

Counted Queue

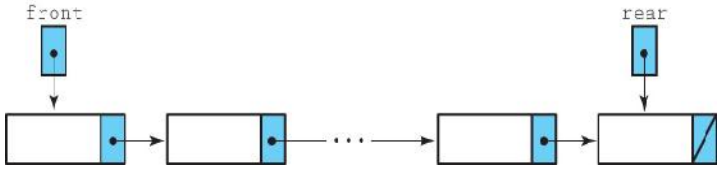
Does the derived class have access to the base class's private data members in C++?

Exercise: finish Counted Queue Implementation.

Linked Implementation

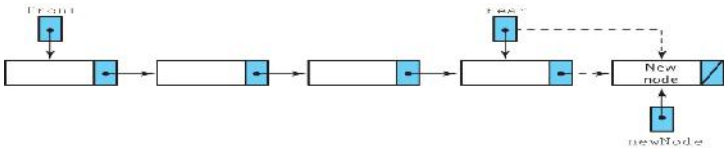
Data structure for linked queue

What data members are needed?



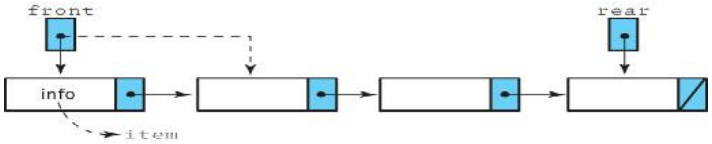
Linked Implementation

Enqueue(newItem)
Set newNode to the address of newly allocated node
Set Info(newNode) to newItem
Set Next(newNode) to NULL
If queue is empty
Set front to newNode
else
Set Next(rear) to newNode
Set rear to newNode

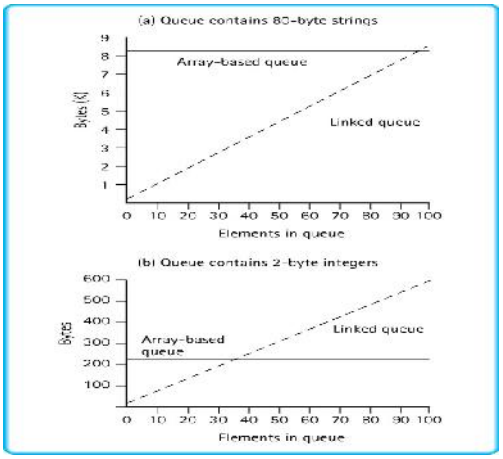


Linked Implementation

Dequeue(item)
Set tempPtr to front
Set item to Info(front)
Set front to Next(front)
if queue is now empty
Set rear to NULL
Deallocate Node(tempPtr)



Storage Requirements



What does this table tell you ?

Reference

- Reproduced from C++ Plus Data Structures, 4th edition by Nell Dale**.**
- **Reproduced by permission of Jones and Bartlett Publishers International.**